Class Grammar:

self.start: str = None # starting nonterminal

self.terminals: list[str] = [] # list of terminals

self.nonterminals: list[str] = [] # list of nonterminals

self.productions: dict[str, list[list[str]]] = {} # dictionary of nonterminal -> list of productions

def \_load(self, filepath: str) -> None:

""" Loads grammar from `filepath`. """

def is\_cfg(self) -> bool:

""" Checks if grammar is context-free. """

class Action:

""" Parent class for all 3 types of actions: shift, reduce and accept"""

Pass

class Accept(Action):

def apply(self, config: Configuration) -> None:

""" Performs accept action on LR(0) configuration. """

class Reduce(Action):

def \_\_init\_\_(self, nonterminal: str, production: list[str], production\_no: int):

""" Initializes a reduce action for nonterminal and its production. """

def apply(self, config: Configuration, table: dict[State, Action]) -> None:

""" Performs reduce action on LR(0) configuration. """

class Shift(Action):

def \_\_init\_\_(self):

self.goto: dict[str, State] = {} //the dictionary for goto

def apply(self, config: Configuration) -> None:

""" Performs shift action on LR(0) configuration. """

class Configuration:

""" A configuration of LR(0) parser. """

class Item:

""" An item of the LR(0) parser. """

nonterminal: str # the nonterminal on the left-hand side production

production: list[str] # the right-hand side of the production

current\_symbol\_idx: int # the symbol of the production that is currently processed (it represents the symbol right after the dot)

class Parser:

""" LR(0) parser. """

def \_\_init\_\_(self, grammar\_path: str | Path):

""" Initializes the parser with the given grammar.

Args:

grammar\_path (str | Path): path to grammar description

"""

def parse(self, pif: ProgramInternalForm) -> ParserOutput:

""" Parses the PIF using the syntax described in `syntax\_path`.

The LR(0) parser is used.

"""

def \_augment\_grammar(cls, grammar: Grammar) -> Grammar:

""" Returns an augmented grammar of `grammar`.

Args:

grammar (Grammar): grammar of the language

Returns: an augmented grammar of `grammar`

"""

def \_get\_closure(self, initial\_item: Item) -> set[Item]:

""" Computes the closure of `initial\_item`.

Args:

initial\_item (Item): the item on which the closure is computed

Returns: the closure as a list of items

"""

def \_goto(self, state: State, symbol: str) -> None | State:

""" Performs the goto action on `state` using `symbol`.

Args:

state (State): the state on which the goto is performed

symbol (str): the symbol used in goto

Returns: the new State if one was found, otherwise None

"""

def \_get\_canonical\_collection(self) -> set[State]:

""" Computes the canonical collection of the grammar.

Returns: the canonical collection as a list of states

"""

def \_generate\_parsing\_table(self, canonical\_collection: set[State]) -> dict[State, Action]:

""" Generates the LR(0) parsing table based on the canonical collection.

Args:

canonical\_collection (set[State]): the canonical collection of the augmented grammar

Returns: the parsing table as a mapping from a State to an Action

"""

def \_find\_next\_state(\_item: Item) -> State | None:

""" Finds the next state based on an item by checking what state contains that item. """

class ParserOutput:

""" Represents the output of parsing """

class Entry:

""" Entry in parsing table. """

symbol: str

parent: int

right\_sibling: int

def \_\_init\_\_(self, grammar: Grammar, output\_stack: Stack):

""" Initializes and constructs the parsing table given the grammar and the string of production found in `output\_stack`.

Args:

grammar (Grammar): grammar of the language

output\_stack (Stack): output stack of LR(0) parser (contains string of productions)

"""

def \_construct(self, grammar: Grammar, output\_stack: Stack) -> list["ParserOutput.Entry"]:

""" Constructs the parsing table given the grammar and the string of production found in `output\_stack`.

Args:

grammar (Grammar): grammar of the language

output\_stack (Stack): output stack of LR(0) parser (contains string of productions)

Returns: the parsing table represented as a list of `ParserOutput.Entry` instances

"""

def dump(self, filepath: str | Path) -> None:

""" Dumps the parsing table to a file.

Args:

filepath (str | Path): path to file

"""

class State:

""" A state of the LR(0) parser. """